Eat All You Can in an All-You-Can-Eat Buffet: A Case for Aggressive Resource usage

Ratul Mahajan Jitendra Padhye Ramya Raghavendra Brian Zill
Microsoft Research

***Abstract* —** In contrast to a focus on efficiency, we advocate aggressive usage of available resources. This view is embodied in what we call the Buffet principle: continue using more resources as long as the marginal cost can be driven lower than the marginal benefit. We illustrate through several examples how this seemingly obvious principle is not adhered to by many common designs and how its application produces better designs. We also discuss broadly the considerations in applying the Buffet principle in practice.

1. INTRODUCTION

Alice walks into a restaurant with an all-you-can-eat buffet. She wants to eat enough to avoid hunger until the next meal. Should she eat based on the expected time until her next meal, or should she eat as much as she can?

The second strategy is clearly superior. It provides the best possible protection against hunger, limited only by the capacity of Alice's stomach. With the first strategy, misestimation of the time of the next meal or of the activity level lead to hunger. And note that both strategies cost the same.

Surprisingly, system design often follows the first strategy today. For instance, consider the task of adding forward error correction (FEC) to transmissions over a wireless channel. In current designs, the number of added FEC bits tends to be a function of the anticipated bit error rate [2, 4, 23, 8], independent of the available spectrum resources. This method protects against packet loss as long as the errors are fewer than anticipated but fails with higher or bursty errors. This failure is unfortunate if there are available resources that would otherwise go to waste.

Underlying the use of the first strategy today is a desire for efficient use of available resources. In the FEC example, adding the number of bits that is a function of the common-case error rate is an efficient way to use the spectrum. More bits might be considered wasteful usage. Yet if that spectrum would otherwise go unused, the real waste is in not taking advantage of it to improve performance. As demonstrated by the examples above, a singular focus on efficiency can lead to poor performance.

Based on these observations, we put forth the *Buffet* principle: continue using more resources as long as the marginal cost can be driven lower than the marginal benefit. Stated differently, efficiency of resource usage should not be a driving concern if more resources can be used at a lower cost than the benefit from the additional use.

Through several case studies, we show that applying the Buffet principle produces designs that are qualitatively different and arguably perform better. Our cases span a range of systems and resource types, including erasure coding over lossy channels, replication for reliability, managing control traffic, and speculative execution. The diversity of these examples points to the broad applicability of the principle.

The key challenge in applying the Buffet principle is that the default way to greedily use resources tends to be costly. For example, in the FEC scenario, if the network is CSMA-based and a transmitter greedily pads its data transmissions, other transmitters will suffer and total network goodput will drop. Unless this challenge can be overcome, efficiency-oriented designs are likely prudent.

Our case studies suggest that this challenge can be met in many settings. In the FEC scenario, for instance, it can be met by having transmitters send additional FEC bits in separate, short transmissions that occur with a lower priority, so that they are less likely to hinder other data transmissions. In addition to prioritization, we identify opportunistic usage when the resource is vacant, utility-driven usage, and background usage as useful methods in building Buffet-based systems.

We also discuss broadly the other challenges in applying the principle, its limitations, and scenarios where it can be naturally applied. These scenarios are where the opportunity cost of greedily using resources can be effectively controlled; where the resource in question goes to waste if not used; and where greedy usage by one user does not hurt others. The potential limitations of Buffet-based designs are that performance can become a function of the amount of spare resources and greedy usage of one resource can increase the latency of certain tasks and bottleneck other resources.

We do not claim that the Buffet principle has never been used before. For example, one recent work appears to use it [6], and there are undoubtedly others as well. In contrast to these works, the contribution of this paper lies in an explicit and general specification of the principle and in provoking a broader discussion of its value. In this respect, we are inspired by the end-to-end argument [16], which articulates a broadly useful principle across the design of many systems.

We also do not claim that the principle can be universally applied, only that it offers a useful perspective on system design. The most attractive aspect is that the per-

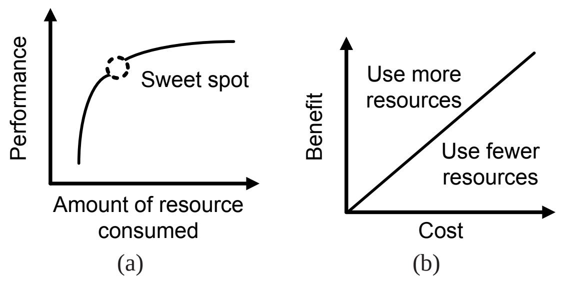

Figure 1: (a): The thinking underlying many efficiency-centric designs. (b): A simplistic illustration of the Buffet principle.

formance of Buffet designs would be limited primarily by the amount of available resources, rather than likely artificial limitations driven by efficiency concerns. However, its full potential can only be understood in the context of concrete, practical designs. We are currently building two different systems based on the Buffet principle.

2. THE (SOMETIMES MISPLACED) FOCUS ON EFFICIENCY

In this section, we describe how the focus on efficiency manifests in system design today and when it may be unwarranted. In many systems, the amount of resources used depends on design choices, rather than it being a simple function of workload. Examples include systems that: $i)$ add FEC to data transmitted over a communication channel, where the amount of resources consumed depends not only on the payload but also on the extent of error correction added; $ii)$ replicate data over multiple storage devices, where the amount of resources consumed depends on the degree of replication; $iii)$ prefetch libraries into memory before user programs ask for it (to speed execution), where the amount of resources used depends on the aggressiveness of prefetching.

To those familiar with the design of such systems, Figure 1(a) may appear familiar. The x-axis represents the amount of resources consumed and the y-axis represents performance. In the FEC case, these can be the number of added bits and the fraction of packets correctly received.

System designers often use such a graph as a guide. They try to find the "sweet spot" such that: $i)$ before it, consuming more resources brings great additional benefit; and $ii)$ beyond it, there are diminishing returns. The sweet spot is an attractive operating point when efficiency, which may be characterized as performance per unit of resource consumed, is a central goal.

However, an exclusive focus on efficiency can be misplaced. We outline specific examples in §4, but the general characteristics of such situations are the following.

- Extra resources can be used such that the marginal cost is low and the resource itself is of "use it or lose it" variety, that is, not using it leads to unnecessary wastage. Such resources include disk space, channel capacity, etc. While the returns from using resources beyond the sweet spot are low, they nevertheless represent additional benefit, which should be had when the cost is low.
- The amount of resource usage needed to hit the sweet spot is hard to determine accurately because the system is dynamic. This occurs, for instance, when the failures or packet losses are bursty; here, even if the view of average failure or loss rate is accurate, burstiness implies that at any given instance the system may be operating far from the sweet spot. The system would perform better and the design may be simpler as well if the focus was on using as much resource as possible, rather than trying to operate at constantly a moving target.

We argue that instead of focusing exclusively on efficiency, the designers must take a holistic look at the resources at their disposal and use them aggressively. Towards this end, we propose the Buffet principle.

3. THE BUFFET PRINCIPLE

The Buffet principle is easily stated: *continue using more resources as long as the marginal cost can be driven lower than the marginal benefit.* Figure 1(b) illustrates it somewhat simplistically, without capturing the dynamics of marginal cost and benefit and thus the fact that the designs may get less efficient as more resources are used.

The simplicity of the Buffet principle is deceptive, to the extent that it might seem obvious and in wide usage. But system design today is often not approached from the perspective advocated by it. This point will be clarified below and in the case studies outlined in the next section.

For a quick illustration, however, consider TCP, the dominant transport protocol for reliable communication. At first glance, it may appear that TCP uses the Buffet principle because it tries to estimate and consume all available bandwidth. However, TCP consumes all available bandwidth only if there is sufficient amount of new data, for instance, during a large file transfer. It will not use the spare bandwidth to proactively protect existing data from loss.

For example, consider the case where TCP's congestion window is 8 packets and it receives only 4 packets from the application. TCP will send only 4 packets even though the path can support more, assuming that congestion window reflects available bandwidth. It will send more only after a packet is determined to be lost, which takes at least a round trip time.

A Buffet-based transport protocol might preemptively send each packet twice, thus using the spare bandwidth to provide faster loss recovery. Of course, whether such a protocol is practical depends on whether other data can be protected from the aggressive bandwidth usage by duplicate packets.

As suggested by the example above, the key to successfully applying the Buffet principle is that the aggressive resource usage advocated by it must be enabled in a way that does not hurt overall performance. Otherwise, the marginal cost would be high and an efficiency-focused design would in fact be prudent. The default way to aggressively use resources often has a high cost; for instance, the duplicate packets above may lead to higher overall loss rate. This reason is perhaps why many system designs tend to focus on efficiency, almost by default; it is not the case that designers are leaving obvious gains on the table.

Approaching the design from the perspective of the Buffet principle challenges designers to devise methods to lower the impact of aggressive resource usage. The examples below highlight that this is likely achievable in many cases. The resulting designs can be qualitatively different, sometimes simpler, and perform better.

Applying the Buffet principle also requires us to quantify or at least compare the cost and benefit of using more resources. This exercise is system-specific and must account for all relevant economic and performance-related factors. We discuss this challenge in §5.1.

4. CASE STUDIES

We now describe several settings that can benefit from Buffet-based designs. We classify them based on the nature of the resource of interest. Our designs are not complete but are meant to highlight the diversity of settings where the principle can be applied. The next section has a more general discussion of considerations surrounding the application of the principle.

4.1 Wireless spectrum or bandwidth

4.1.1 Forward error correction (FEC)

Wireless media tends to be error-prone and the bits inferred by the receiver may be corrupted in transmission. Adding FEC bits can help recover from some of the bit errors and improve performance by reducing packet loss. The trade-off here is that each additional bit can lower packet loss but also steal transmission capacity.

FEC designs that we are aware of either add a fixed number of bits to each transmission or a number that adapts based on estimated bit error rate (BER) [2, 4, 23, 8]. Current designs use efficiency arguments similar to those in §2 and add bits corresponding to the sweet spot where additional bits present a diminishing reduction in loss rate. However, by not explicitly considering available resources, they either unnecessarily lose packets even when there are spare resources or create unnecessarily high FEC overhead under heavy load. Either way, throughput suffers.

A Buffet-based FEC design can enable the maximal protection against bit errors that the amount of available spectrum resources can provide. Such a design will add some minimum number of FEC bits to all transmissions, perhaps based on the expected common case BER. On top of that, it will greedily add more FEC bits as long as there are spare resources.

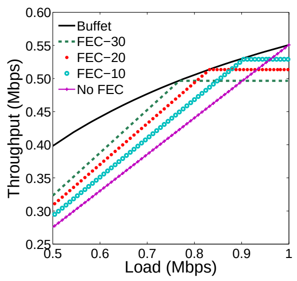

Figure 2: Throughput with different FEC mechanisms as a function of offered load.

We illustrate the benefits of such a design using a simple simulation experiment in which 8000-bit packets are sent over a channel with 1 Mbps capacity and a BER of 10^{-6}. We assume an optimal FEC code: when k bits of FEC are added, the packet is successfully delivered if any 8000 out of 8000+k bits are received without error. Figure 2 shows the throughput in this setting without FEC, with different levels of added FEC, and with a Buffet design where the minimum number of FEC bits is zero. FEC-K refers to adding FEC bits equivalent to K% of the packet size – current FEC designs would sit on one such curves. We see that the Buffet-based FEC performs the best across the board. For any given load level, the Buffet-based design matches the best other design. Individual other designs suffer significantly either under low load or under high load.

The example above also suggests how Buffet designs can be simpler. Current FEC designs need to carefully decide how many bits to add based on estimated BER or packet losses [2, 4, 23]. This task is complicated by the bursty and dynamic nature of the error process, and misestimations hurt throughput. Buffet designs skirt this complexity altogether. By simply adding as many bits as the currently available resources allow, they can get the best performance at all load and BER levels.

A challenge, however, in the design of a Buffet-based FEC system is to ensure that greedy addition of FEC bits does not lead to fewer data bits being transmitted (e.g., due to carrier sensing). This property is easy to achieve in systems where transmitters have a short- or long-term dedicated share of the medium, as may be the case for satellite links or long-distance point-to-point links [8, 14].

It can also be achieved in CSMA-based systems. Instead of embedding all FEC bits in the data packet itself, we can embed the minimum number of required bits in the packet. The additional bits are transmitted separately with lower priority, which makes it more likely for data transmissions of other senders to acquire the medium. Such priority mechanisms can be implemented today using recent WiFi hardware that supports quality of service (QoS) enhancements (802.11e) [1]. We can further reduce the impact of greedy FEC bits by making FEC-only packets small, so that even when they do acquire the medium, they delay data transmissions by only a short amount of time.

We are currently designing such an FEC system. Our focus is on VoIP and multimedia streaming. For these applications, the aggressive addition of FEC bits would lead to more timely data delivery, compared to retransmissions based on exponential backoffs.

4.1.2 Erasure coding for lossy paths

Rationale similar to the one above also applies to protection against packet losses. For this setting as well, current designs can lead to avoidable data loss. As an example, consider a recent system, called Maelstrom [5], that uses erasure coding to combat losses in dirty fiber. It adds a fixed amount of redundancy to the data stream, based on the observation that loss rates in fiber are generally low and adding more redundancy would use more resources in the common case. With Maelstrom, data would be lost whenever loss rate is higher than the level of protection. A Buffet-based system can provide greater protection from losses by utilizing all remaining path capacity for erasure coded packets.

The key challenge here is to send coded packets such that they do not steal bandwidth from normal data traffic. This is easily accomplished in a system like Maelstrom if sits on the two ends of the fiber. It can also be accomplished by marking redundant information as lower priority, so that routers drop them first during periods of congestion. A way to accomplish it without router support is to send erasure coded data opportunistically, only when the queues are estimated to be empty.

We are building a system that uses the third method above. It targets paths provided by cellular providers from moving vehicles; such paths tend to be lossy with unpredictable loss rates [15]. Their roughly stable capacity lets us estimate when the queues are empty and erasure coded packets can be sent. This system is meant for users that subscribe to an unlimited data plan, and thus the marginal cost of sending erasure coded data is only performance-related, not economic. Our early experiments show a negligible drop in throughput due to aggressive coding, even under high offered load. They also show an appreciable reduction in packet losses.

4.1.3 Mobility updates

The performance of systems that exhibit a high-degree of mobility, such as a mobile ad hoc network (MANET), depends on the frequency of mobility updates. A higher frequency yields better performance as nodes will have a more up-to-date view of the topology, but it can also swamp data traffic. Existing systems get around this trade-off by setting the frequency of updates to a tolerable level that is based on an analysis similar to the sweet spot reasoning presented in the previous section [10, 3]. Such systems may perform poorly in situations with higher than anticipated mobility levels even when there is spare capacity to support a high update frequency.

A Buffet-based mobility update mechanism will provide better performance whenever spare capacity is available. The practical difficulty here again is ensuring that the additional updates do not hurt data traffic. This can be accomplished using a priority mechanism similar to the one suggested above for FEC transmissions.

4.1.4 Routing in delay tolerant networks (DTNs)

As further evidence of the value of the Buffet principle, we note that system design in the domain of DTN routing has evolved from not using the principle to using it. Many DTN routing protocols replicate messages along multiple paths to improve their delivery probability. Older protocols limit the amount of replication to prevent a few messages from dominating network resources [17, 12, 18]. Because this limit is not guided by the amount of available storage or bandwidth between replication end points, these designs can perform poorly even when plenty of resources are available. A recent protocol, called RAPID [6], implicitly uses the Buffet principle. It replicates as much as available resources allow. To prevent network domination by a few messages, it takes a utility-driven approach in which messages are replicated based on their expected utility. Messages that have been replicated more have lower utility. The authors demonstrate that RAPID significantly outperforms older designs.

4.2 Storage

4.2.1 Long-term storage

Replication protects data against node failures and latent sector errors in disks. The amount of replication, however, is often pre-determined today, based on anticipated failure rate. This unnecessarily limits the protection level even when there may be spare resources. A replication system based on the Buffet principle will provide maximal protection given available resources.

Consider two scenarios. The first is replication across one or more disks on a single computer. Today's mechanisms such as various RAID configurations are based on a preset amount of replication that provides protection against a certain number of failures. This can lead to data loss when more failures occur even though ample working storage may still be available. A Buffet-based design will replicate aggressively to fill all available storage, thus providing maximal possible protection. The key challenge is to not hurt read and write performance in the process, which we believe can be accomplished by relegating the task of additional replication to the background and conducting it only when the disk is idle.

The second scenario is replication across computers in a data center or in a wide-area peer-to-peer system. Here too, the system will be more reliable with replication that uses all available resources rather than a fixed replication level. The key challenge is to manage the bandwidth impact of aggressive replication, which is a particularly relevant concern for the wide-area setting. We believe that this concern can be handled through background transfer protocols such as TCP Nice [21].

4.2.2 *Short-term storage*

Program execution can be slowed by the time it takes to load the program and the libraries it uses into memory. One can speed this up by preemptively loading in memory commonly used binaries when there is space (and time) available. Indeed, this strategy has already been proposed or implemented for modern operating systems [9, 19]. A Buffet-based strategy will maximize performance by aggressively filling available memory, instead of being limited to the most promising candidates.

Similar ideas have been explored in the context of prefetching web pages that users are likely to view in the future [11, 22, 13]. If the bandwidth impact of such prefetching can be controlled, for instance, using TCP Nice, such systems should aggressively fill available cache capacity to maximize user-perceived performance.

4.3 Computational resources

Speculative execution is a commonly used technique in modern processors. In it, parts of code are executed even though the results may eventually be discarded, depending on the outcome of the (if) conditions that occur prior to these parts. The execution of the program is non-sequential to parallelize processing. When the results prove useful, speculative execution boosts performance. The performance benefit of speculative execution depends on the accuracy of branch prediction. Conventionally, only one branch is speculatively executed even though additional resources may be available for executing more branches. More recent designs attempt to execute multiple paths [20]. For maximal performance, a Buffet design would speculatively follow as many paths as current resources levels allow. As the number of cores inside processors increase, such a design would increasingly outperform strategies that limit speculative execution to more likely paths.

5. APPLICABILITY CONSIDERATIONS

In this section, we discuss broadly the issues related to applying the Buffet principle in practice. These are based on our early experiences and will be refined over time.

5.1 Challenges in applying the principle

There are two key challenges. The first challenge of course is ensuring that greedy resource usage does not detract from other productive work. The last section mentions several techniques to address this challenge in the context of specific examples. We summarize them here. One technique is prioritization, so that greedy tasks get lower priority. Prioritization can be explicit, e.g., embedding priority in packet headers for routers. It can also be implicitly implemented by sources, by them deferring to other tasks, e.g., background transfers of TCP Nice [21] and the use of higher inter-frame spacings in 802.11e [1]. Prioritization may not suffice in settings where aggressive usage of multiple nodes need to be traded-off with one another based on their relative benefit. Utility-driven resource consumption, which is a generalization of prioritization, can help here. In it, tasks are executed in order of their estimated utility, as in RAPID [6]. Yet another technique is opportunistic usage, as in our erasure coding system (§4.1.2) in which greedy usage occurs only when the resource is idle. We believe that one of these techniques or a combination can be applied in many situations.

The second challenge is quantifying or at least being able to compare the marginal benefit and cost of using more resources. For cost, the primary difficulty is taking into account the opportunity cost of greedily using resources, that is, for what else could those resources be used. This is not a concern where the greedily allocated resource can be easily reclaimed when needed or would otherwise remain unused. But it could be problematic otherwise. Additionally, if precise accounting is desired, we need to quantify the cost of the side-effects produced by greedy usage as well (§5.3).

We can avoid the task of quantifying marginal cost by driving it to zero or negligible levels. The techniques above for managing greedy usage help here. If done successfully, we can continue to use more more resources until the marginal benefit becomes negative.

Quantifying marginal benefit can also be tricky, e.g., in the face of correlated failures [7]. But because the marginal benefit of using more resources is usually positive, more resources can be used whenever the marginal cost is negligible.

5.2 Applicable resources

Two categories of resources are well-suited for applying the Buffet principle. The first is non-conservable resources, i.e., those that would go to waste if not used. Storage, bandwidth, and computational resources are typically non-conservable. An example of a conservable resource is battery power.

We do not claim that the Buffet principle does not apply to conservable resources, only that it is easily applied to non-conservable resources. Applying it to conservable resources requires a more involved evaluation of marginal benefit that includes predictions of future behavior.

The second category is where the resource is not shared with non-Buffet users who may not be able to differentiate normal usage from greedy usage with lower value. Such users might reduce their own consumption on observing aggressive usage, which would reduce overall system throughput. In some cases, Buffet users can co-exist with non-Buffet users. For instance, our wireless FEC design co-exists by implementing greedy usage at lower priority and deferring to non-Buffet users.

5.3 Side effects of greedily using resources

We have encountered three side-effects. First, the system performance becomes a function of the workload itself. For example, our FEC design loses fewer packets at lower loads and more at higher loads; in current designs, the loss rate for transmitted packets is independent of load. One might argue that such load-dependent performance abstraction is hard for applications. But observe that performance is already often load-dependent. For instance, wireless loss rate increases with load because the collision frequency increases. Even along wired paths, loss rate observed by applications can depend on load. Sending at 0.9 Mbps along a 1 Mbps capacity path leads to no loss but sending at 1.1 Mbps leads to 10% loss.

The second side-effect is that greedy usage can strain the system. It can increase task-completion latency. For instance, a read request for a disk block will have to wait longer if it arrives during greedy replication. The level of latency increase depends on how fast the greedy task can be completed or preempted. It can be controlled by keeping individual greedy tasks short or preemptable. Another strain is that aspects of the system that were originally not the bottleneck can become bottlenecks with greedy usage. For instance, disk I/O bandwidth may become a bottleneck with aggressive replication, even if it was not previously. Careful design is needed to alleviate this problem.

A final side-effect is that with greedy usage, the resources will frequently appear fully utilized. This behavior will typically not matter but it may in some cases, for instance, if administrators use utilization levels to make provisioning decisions. This issue can be dealt with by separately counting normal and greedy usage.

5.4 Benefit of the principle in practice

It depends on the workload and the amount of available resources. So it might vary from none to a lot. For example, in our erasure coding system, a Buffet-based design leads to zero loss under low load and a a loss rate that is equal to the underlying path loss rate under heavy load. The appropriate view of a Buffet design is that its performance is limited by the amount of spare resources instead of specific design parameters, and thus it provides the best performance for a given level of resource investment.

6. CONCLUSIONS

We articulated the Buffet principle, which advocates a different perspective on system design than a singular focus on efficiency. Through several examples, we explain how Buffet designs differ from efficiency-centric designs and how they are likely to perform much better. We also discussed broadly the considerations surrounding the application of the principle in practice. This discussion points to both strengths as well as limitations. Overall, we find the principle promising and offering a useful perspective on system design. Instead of being limited by artificial design choices, Buffet designs have the potential to provide the best performance for the level of available resources. Its eventual worth can be understood only by studying the performance of many concrete designs, which is an active area of research for us.

Acknowledgments We thank John Douceur for feedback on this paper.

7. REFERENCES

[1] Medium access control (MAC) quality of service enhancements. IEEE Standard, 2005.

[2] J.-S. Ahn, S.-W. Hong, and J. Heidemann. An adaptive FEC code control algorithm for mobile wireless sensor networks. *Journal of Communications and Networks*, 7(4), 2005.

[3] S. Ahn and A. Shankar. Adapting to route-demand and mobility in ad hoc network routing. *Computer Networks*, 38(6), 2002.

[4] A.Levisianou, C.Assimakopoulos, N-F.Pavlidou, and A.Polydoros. A recursive IR protocol for multi-carrier communications. In *Int. OFDM Workshop*, Sept. 2001.

[5] M. Balakrishnan, T. Marian, K. Birman, H. Weatherspoon, and E. Vollset. Maelstrom: Transparent error correction for lambda networks. In *NSDI*, Apr. 2008.

[6] A. Balasubramanian, B. Levine, and A. Venkataramani. DTN routing as a resource allocation problem. In *SIGCOMM*, Aug. 2007.

[7] J. R. Douceur and R. P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *MASCOTS*, Aug. 2001.

[8] M. Emmelmann and H. Bischl. An adaptive MAC layer protocol for ATM-based LEO satellite networks. In *VTC*, Oct. 2003.

[9] B. Esfabd. *Preload: An adaptive prefetching daemon*. PhD thesis, University of Toronto, 2006.

[10] R. K. Guha, Y. Ling, and W. Chen. A light-weight location-aware position update scheme for high mobility networks. In *MILCOM*, Oct. 2007.

[11] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE JSAC*, 16(3), 1998.

[12] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *SAPIR*, Aug. 2004.

[13] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *SIGCOMM*, Aug. 1996.

[14] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer. WiLDNet: Design and implementation of high-performance WiFi-based long distance networks. In *NSDI*, Apr. 2007.

[15] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. MAR: A commuter router infrastructure for the mobile Internet. In *MobiSys*, June 2004.

[16] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM ToCS*, 2(4), 1984.

[17] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN*, Aug. 2005.

[18] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Performance analysis of mobility-assisted routing. In *MobiHoc*, May 2006.

[19] Windows Vista SuperFetch. http://www.microsoft.com/windows/products/windowsvista/features/details/superfetch.mspx.

[20] A. K. Uht. *Speculative Execution in High Performance Computer Architectures*, chapter Multipath Execution. CRC Press, 2005.

[21] A. Venkataramani, R. Kokku, and M. Dahlin. TCP-Nice: A mechanism for background transfers. In *OSDI*, Dec. 2002.

[22] Q. Yang and H. H. Zhang. Integrated web prefetching and caching using prediction models. *WWW*, 4(4), 2001.

[23] L. Zhao, J. W. Mark, and Y. Yoon. A combined link adaptation and incremental redundancy protocol forenhanced data transmission. In *GLOBECOM*, Nov. 2001.